A Residue Number System on Reconfigurable Mesh with Applications to Prefix Sums and Approximate String Matching

Alan A. Bertossi and Alessandro Mei

Abstract—Several new number representations based on a Residue Number System are presented which use the smallest prime numbers as moduli and are suited for parallel computations on a reconfigurable mesh architecture. The bit model of linear reconfigurable mesh with exclusive write and unit-time delay for broadcasting on a subbus is assumed. It is shown how to convert in $O(1)$ time any integer, ranging between 0 and $n-1$, from any commonly used representation to any new representation proposed in this paper (and vice versa) using an $n \times O\Big(\frac{\log^2 n}{\log\log n}\Big)$ reconfigurable mesh. In particular, some of the previously known conversion techniques are improved. Moreover, as a byproduct, it is shown how to compute in $O(1)$ time the Prefix Sums of n bits by a reconfigurable mesh having the above mentioned size, thus improving previously known results. Applications to the Prefix Sums of N h-bit integers and to Approximate String Matching with α mismatches are also considered. The Summation and the Prefix Sums can be computed in $O(1)$ time using $O\Big(h\log N + \frac{\log^2 N}{\log\log N}\Big) \times Nh$ and $O\Big(\frac{h^2+\log^2 N}{\log(h+\log N)}\Big) \times O(N(h+\log N))$ reconfigurable meshes, respectively. Moreover, it is shown for the first time how to find in $O(1)$ time all the occurrences of a pattern of length m in a text of length n, allowing less than α mismatches, using a reconfigurable mesh of size $O(m\log|\Sigma|) \times O\Big(n\Big(\log|\Sigma| + \frac{\log^2\alpha}{\log\log\alpha}\Big)\Big)$, where the pattern and the text are strings over a finite alphabet Σ and $\alpha < m \leq n$.

Index Terms—Number representation, prefix sums, reconfigurable mesh, residue number system, string matching with k mismatches, VLSI.

1 Introduction

In the last decade, reconfigurable meshes have been extensively studied due to both their reconfiguration capabilities, which permit a fast communication among processors, and to technological developments, which allowed some experimental and commercial reconfigurable chips with thousands of processors to be built [1], [14].

In a *reconfigurable mesh*, each node can dynamically connect and disconnect its adjacent edges in various patterns. Specifically, each *node* of the mesh consists of a *processing unit*, a small *local memory*, and a *switch*, while each *edge* is viewed as a building block for a larger *bus*. Each switch has some I/O ports and each port is directly connected to at most one edge. While the edges outside the switch are fixed, the internal connections between the I/O ports of each switch can be locally configured by the processor itself into any partition of the ports. In this way, during the execution of an algorithm, the edges of the mesh are dynamically partitioned into edge-disjoint *subgraphs*. Every such subgraph forms a *subbus* and allows the processors of the subbus to broadcast, at any given time, a message to all the other processors sharing the same subbus.

Several variants of the model described above have been defined, depending on the kind of allowed partitions, subbus arbitration, and local memory size. Two main models are usually considered with respect to the allowed partitions:

- *General Reconfigurable Mesh*. Any partition of the ports is allowed. Thus, the possible configurations are any mesh partition in edge-disjoint connected subgraphs.
- *Linear Reconfigurable Mesh*. Only partitions made of pairs and singletons are allowed. In this way, each subbus is of the form of either a path or of a cycle.

Moreover, two main models are considered with respect to the kind of subbus arbitration:

- *Exclusive-Write*. Only one processor is allowed to broadcast on a subbus at any single step of computation.
- *Common-Write*. More processors are allowed to broadcast on the same subbus during the same step, provided that all of them broadcast the same value which in turn consists in a single bit only.

Finally, two models are also considered with respect to both the size of each local memory and the width of the busses: the bit model and the word model, where the above parameters are either both constant or both logarithmic in the size of the mesh, respectively. In this paper, we focus on the Exclusive-Write, Linear Reconfigurable Mesh bit model, that is clearly the weakest among the above mentioned

- *The authors are with the Department of Mathematics, University of Trento, 38050 Povo (Trento), Italy.*
E-mail: {bertossi, mei}@science.unitn.it.

Manuscript received 23 Apr. 1998; revised 17 Mar. 2000; accepted 14 July 2000.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 106753.

models. Thus, the algorithms to be proposed in this paper could also be executed on whichever of the other models with no loss in performance.

Efficient parallel algorithms that use reconfigurable networks have been devised for many problems, such as sorting [6], [10], [13], [17], [21], matrix and integer multiplication [19], [9], bit and integer summation [11], [18], string and pattern matching [5], [7], finding the connected components of a graph [14], image processing [12], [14], parallel data structures [2], and dynamic programming [3]. Most of such algorithms achieve an $O(1)$ time complexity by considering the so-called *unit-time delay* model, in which it is assumed that each broadcast takes constant time to reach all the processors in a subbus.

An aspect of paramount importance for designing efficient parallel algorithms on reconfigurable meshes consists of the methods used to represent numbers and to do conversions between different number representations. Indeed, although input and output data are usually represented using the classical binary representation, unary representations are extensively used for data processing within the mesh in order to reach constant processing times. Moreover, more sophisticated representations based on Residue Number Systems (RNS) could also be used since they are suited for exploiting the reconfiguration capabilities of the mesh [19]. In an RNS, an integer is represented by a tuple of elements, where each element is the remainder of the division of the represented number by another number called modulus and, thus, many operations can be easily parallelized by performing them separately on each remainder. By the Chinese Remainder Theorem, when the moduli are pairwise relatively prime, the RNS representation of a number is unique. Using a reconfigurable mesh, the moduli have to be carefully chosen in order to minimize the sum of the moduli themselves and, thus, the size of the RNS representation, while maximizing their product and, thus, the greatest representable number. As suggested in [15], the best choice consists of selecting the smallest prime numbers as moduli.

In this paper, several new RNS number representations are presented which indeed use the smallest prime numbers as moduli. In particular, it is shown how to convert, in $O(1)$ time, any integer, ranging between zero and $n-1$, from any commonly used binary or unary representation to any new binary or unary representation proposed in this paper (and vice versa) using an $n \times O(\frac{\log^2 n}{\log\log n})$ reconfigurable mesh. As byproducts, some of the previously known conversion techniques are improved and a new algorithm for computing in $O(1)$ time the Prefix Sums of n bits is proposed. Given n bits $y_0, \dots, y_{n-1}$, the Prefix Sums problem consists of computing $z_0, ..., z_{n-1}$, where $z_i = y_0 + \dots + y_i$, for $i = 0, ..., n-1$. The new algorithm for the Prefix Sums works on a reconfigurable mesh having a $2n \times O(\frac{\log^2 n}{\log\log n})$ size, and improves all the previously known algorithms (indeed, an $O(1)$ time algorithm on a reconfigurable mesh of the same size was known until now only for the word model [15], but not for the bit model).

Moreover, the RNS representations and conversion techniques introduced in the present paper are applied to compute the Summation and the Prefix Sums of a large number of integers and to solve the Approximate String Matching with α mismatches, where it is assumed that the input and output data are represented by the usual binary notation, while internal data conversions into (and from) the RNS representation are made for achieving a better performance. In particular, it is shown how to compute in $O(1)$ time the Summation and Prefix Sums of N h-bit integers using $O\left(h \log N + \frac{\log^2 N}{\log\log N}\right) \times Nh$ and $O\left(\frac{h^2 + \log^2 N}{\log(h + \log N)}\right) \times O(N(h + \log N))$ reconfigurable meshes, respectively, improving all previously known results for most values of h. Indeed, the best algorithms for the Summation on the bit model are those presented in [11] and [16], which require $O(1)$ time using meshes of size $\min\{N, h \log^2 N\} \times Nh$ and $\sqrt{N} \log^{(O(1))} N \times \sqrt{N}h$, respectively. Therefore, our algorithm is better than that of [11] when $h = o(\frac{N}{\log N})$, while it has a worse product time $\times$ size than that of [16]. However, in contrast to the algorithm proposed in the present paper, the algorithm of [16] is not capable of reading the input and furnishing the output from the boundaries of the mesh in constant time since it requires $\Omega\left(\min\left\{\sqrt{N}, \frac{h\sqrt{N}}{\log^{(O(1))} N}\right\}\right)$ time for I/O. As for the Prefix Sums, no constant time algorithm was devised before for the bit model, the only constant time being that proposed in [18], which uses an $N \times N$ word model of reconfigurable mesh and works only when $h = O(\log N)$. Thus, the constant time Prefix Sums algorithm proposed in the present paper not only is the first such algorithm for the bit model, but also is better than that of [18] since it requires a mesh of smaller size and works for any h.

Finally, it is shown for the first time how to find in $O(1)$ time all the occurrences of a pattern of length m in a text of length n, allowing less than α mismatches, using a reconfigurable mesh of size $O(m \log |\Sigma|) \times O\left(n\left(\log |\Sigma| + \frac{\log^2 \alpha}{\log\log \alpha}\right)\right)$, where the pattern and the text are strings over the finite alphabet Σ and $\alpha < m \leq n$.

The rest of this paper is organized as follows: Section 2 formally defines the model of reconfigurable mesh used throughout this paper, reviews some basic techniques used for computing on reconfigurable meshes, and recalls some notations used in Number Theory. Section 3 is the core of this paper and deals with both old and new number representations, as well as efficient conversion techniques between any pair of representations.

In Section 3.1, the definitions of POS, 1UN, 2UN, and BIN representations given in [10] are reviewed. The first three are unary representations. Even if they need an exponential amount of area to be stored, they are useful to exploit the reconfiguration capabilities of reconfigurable meshes. Moreover, a table look-up technique is introduced, which is extensively used in the rest of this paper. Then, in Section 3.2, the new RPOS, R1UN, R2UN, and RBIN

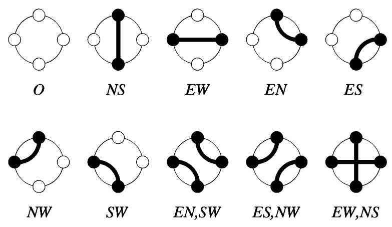


Fig. 1. All the configurations allowed in a switch of a linear reconfigurable mesh.

representations are defined. All of them are RNS representations and the first three of them combine compactness and suitability for reconfigurable computing. Moreover, thanks to these new RNS representations, some of the conversion techniques presented in [9] are improved.

In Section 3.3 the results shown by Frank [8] and concerning disjoint paths on a knock-knee model of rectilinear grid are considered. Due to the importance of unary representations in the field of reconfigurable computing, Frank's results are interpreted as techniques for computing injective functions using the POS representation.

In Section 3.4, the techniques for all the possible conversions between representations are shown. In particular, the BIN $\rightarrow$ RPOS $\rightarrow$ BIN conversion is of paramount importance. Indeed, input and output are often required to be represented in the usual BIN representation. Moreover, the RPOS representation combines good compactness, the ability to exploit the reconfiguring capabilities of the architecture used (due to its unary nature), and adaptedness to some parallelization purposes (due to the remainder number system). As a byproduct, an algorithm for the prefix sums of N bits is given that outperforms all the previously known algorithms.

In Section 4, the problem of adding a large number N of h-bit integers is considered. By a standard parallelization technique that can be used to approach a large class of problems, very efficient algorithms are given for the Summation and the Prefix Sums.

Finally, in Section 5, an algorithm for the well-known problem of Approximate String Matching with α mismatches is presented. To our knowledge, this problem has not been considered before on a reconfigurable mesh. Together with the Summation algorithm and the two Prefix Sums algorithms, the Approximate String Matching algorithm represents a good applicative example of the RNS representations and conversion techniques introduced in this paper.

2 Definitions and Notations

A Linear Reconfigurable Mesh of size $r \times c$ consists of a classical $r \times c$ processor array with additional reconfigurable capabilities [1], [14], [20]. Specifically, there are r rows and c columns of nodes, with edges connecting each node to its four neighbors (or fewer, for borderline nodes). The node in row i and column j is denoted with $P_{i,j}$. Each edge is a building block for a larger bus, while each node has a switch with four I/O ports (*E, W, N, S*), which can be configured in 10 possible ways, as shown in Fig. 1. Each node also has a processor capable of performing the basic arithmetic and logic operations and a small local memory. Processors operate in a single-instruction multiple-data (SIMD) mode and only one processor can broadcast at any time to a subbus shared by multiple processors.

The *bit model* is assumed, where each local memory has a size of $\Theta(1)$ bits and each bus can carry $\Theta(1)$ bits of data. Note that this implies that the processors do not know their indices in the array. It is also assumed that switch reconfiguration can be done in $O(1)$ time by local decisions taken by the processors themselves. It is worth noting that, under the assumptions that processors, switches, and edges occupy $O(1)$ space, the reconfigurable mesh can be laid out on a rectangle of $O(rc)$ area in the VLSI grid model [14].

The literature on reconfigurable meshes has been growing in the last decade, therefore, some main basic techniques are well established today. In particular, it was proven that a $1 \times n$ reconfigurable mesh (e.g., a row or a column of an $n \times n$ reconfigurable mesh) can perform in constant time an AND operation on n Boolean values stored one per processor (see [14], for example). Clearly, the same holds for OR, NAND, and NOR. Moreover, if all the switches of an $r \times c$ reconfigurable mesh configure themselves to EW, r *row buses* are built and these buses can be used to broadcast the content of a column to all the other ones. Similarly, c *column buses* can be built with the NS configuration. All these simple techniques are assumed to be known by the reader.

We end this section by recalling some useful notations. An integer denoted by p is always a prime number, $\log$ denotes the logarithm to the base 2, and we assume the reader is familiar with the standard notation used in Number Theory, which is now briefly recalled.

Let m be a positive integer greater than one. Given two integers a_1 and a_2, the remainder of the division of a_1 by m is denoted by $a_1 \bmod m$. Moreover, we say that $a_1 \equiv a_2 \bmod m$ if there exists an integer q such that $a_1 = qm + a_2$. $\mathbb{Z}/m\mathbb{Z}$ is the quotient set with respect to the relation "$\equiv \bmod m$." This set can be put in bijective relation with $\{0, \ldots, m-1\}$ in a natural way and we will often refer to these two sets as they were the same mathematical object. $(\mathbb{Z}/m\mathbb{Z})^*$ is the set of elements $z \in \mathbb{Z}/m\mathbb{Z}$ for which there exists an element $z^{-1} \in \mathbb{Z}/m\mathbb{Z}$ such that $zz^{-1} \equiv 1 \bmod m$.

If m is a prime number, then $(\mathbb{Z}/m\mathbb{Z})^*$ is equal to $\mathbb{Z}/m\mathbb{Z} \setminus \{0\}$ and there exists an element $c \in \mathbb{Z}/m\mathbb{Z}$ such that

$$\left\{c^i \bmod m \mid i = 0, \ldots, m-2\right\} = (\mathbb{Z}/m\mathbb{Z})^*.$$

A number c with such a property is called a *generator* of $(\mathbb{Z}/m\mathbb{Z})^*$.

3 Discrete Arithmetic on Reconfigurable Mesh

This section deals with known binary and unary representations and new RNS representations, as well as with efficient conversion techniques between pairs of representations.

3.1 Number Representation

One of the key contributions of [10] consists of an organized view of the various representations of integer numbers on reconfigurable models, which are now briefly recalled, together with some algorithms for performing basic arithmetic operations.

Definition 3.1 ([10]). *Given an integer $a \in [0, n-1]$, we define the following representations of a:*

[POS:] n bits $(b_0, b_1, \ldots, b_{n-1})$ are used, where $b_a = 1$ and $b_i = 0$ for all $i \neq a$;

[1UN:] n bits $(b_0, b_1, \ldots, b_{n-1})$ are used, where $b_i = 1$ for all $i \leq a$ and $b_i = 0$ for all $i > a$;

[2UN:] n bits $(b_0, b_1, \ldots, b_{n-1})$ are used, where a bits are equal to one and $n - a$ bits are equal to zero (note that this representation is not unique for a given a);

[BIN:] $\lceil \log n \rceil$ bits $(b_0, b_1, \ldots, b_{\lceil \log n \rceil - 1})$ are used, where $a = \sum_{i=0}^{\lceil \log n \rceil - 1} b_i 2^i$.

In the remainder of this paper, we will often use the strict correlation between the POS and 1UN representations. Thus, it is useful to recall a simple result shown in [10].

Lemma 3.2. *Given the POS (1UN) representation of an integer $a \in [0, \ldots, n-1]$, a $1 \times n$ reconfigurable mesh can compute its 1UN (POS) representation in $O(1)$ time.*

It is clear that the POS, 1UN, and 2UN representations need an exponential amount of area to be stored, but they are interesting since they allow efficient techniques to be used which exploit the reconfiguration capabilities of the mesh. The BIN representation does not seem to have the same capabilities, but it is compact. Our effort is aimed at defining a representation that combines the good qualities of all BIN, POS, 1UN, and 2UN.

In [10], conversion techniques are shown between any possible pair of representations, using the word model of reconfigurable mesh. Moreover, [10] shows how to convert the BIN representation of an integer into its 1UN representation using the bit model. This result can be improved, as will be shown in Corollary 3.4.

In the remainder of the paper, we will often use a simple look-up table technique to compute functions on the POS representation of integers. Therefore, it is useful to give a lemma that shows how these functions are computed.

Lemma 3.3. *Let $f : [0, n-1] \rightarrow [0, m-1]$ be a fixed function. Given the POS representation of $a \in [0, n-1]$ in a row, a $\lceil \log m \rceil \times n$ reconfigurable mesh can compute in $O(1)$ time the BIN representation of $f(a)$. Moreover, given the BIN representation of $b \in [0, m-1]$ in a column, the POS representations of the set $f^{-1}(b)$ can be computed in $O(1)$ time.*

Proof. The BIN representation of $f(j)$ can be stored in the jth column of the mesh, for all $0 \leq j < n$. By broadcasting the POS representation of a to all the rows of the mesh, $f(a)$ can be selected and output, if needed. Indeed, column a is the only column having the bit set to one in the POS representation of a.

Now, suppose that the mesh stores the BIN representation of b in a column. By broadcasting this value to all the columns of the mesh, each column j can check whether b is equal to $f(j)$. In this case, the column outputs a 1 on the first row and the required set is computed. Note that the set may be empty, if f is not surjective, or contain more than one element, if f is not injective. □

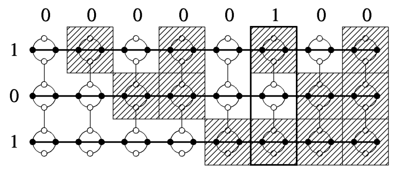


Fig. 2. A 3×8 reconfigurable mesh to convert the POS representation of five into its BIN representation. $P_{i,j}$ is shaded if and only if the ith bit of $f(j)$ is equal to one, where f is the identity function.

The paper [16] gives constant time algorithms to perform the POS $\rightarrow$ BIN and the BIN $\rightarrow$ POS conversions on a $\lceil \log n \rceil \times n$ and $n \times n$ bit model of reconfigurable mesh, respectively. However, the BIN $\rightarrow$ POS conversion can be improved, as shown in the following simple corollary of Lemma 3.3.

Corollary 3.4. *Given the POS (1UN) representation of an integer $a \in [0, n-1]$ in a row, a can be converted into its BIN representation in $O(1)$ time on a $\lceil \log n \rceil \times n$ reconfigurable mesh and vice versa.*

Proof. The proof follows by Lemma 3.3, letting $m = n$, and f equal to the identity function (see Fig. 2). Note that the identity function is obviously injective and surjective, thus the set consists in a single element. If the 1UN representation is used, Lemma 3.2 can be used to get the POS representation. □

The next two lemmas, 3.5 and 3.6, show simple ways to perform arithmetic operations with the POS representations of integers. The proof of Lemma 3.5 is only sketched since it involves a simple application of well-known techniques. Lemma 3.6 provides a multiplication algorithm between integers in $[0, p-1]$, exploiting the primality of p.

Lemma 3.5. *Given the POS representation of two integers $a_1, a_2 \in [0, n-1]$ in a row, the POS representation of*

$$a_1 + a_2 \quad \bmod n \tag{1}$$
$$-(a_2) \quad \bmod n \tag{2}$$
$$a_1 - a_2 \quad \bmod n \tag{3}$$

can be computed in $O(1)$ time on a $\lceil \log n \rceil \times n$ reconfigurable mesh.

Proof. By Corollary 3.4, the BIN representations of a_1 and a_2 can be obtained. Then, (1), (2), and (3) can be computed by standard techniques (see [4], for example). Note that the modulo n operation reduces to adding or subtracting n in this case. □

Lemma 3.6. *Given the POS representations of two integers $a_1, a_2 \in [0, p-1]$ in a row, where p is a prime number and $a_2 \neq 0$, the POS representation of*

$$a_1 \times a_2 \quad \bmod p \tag{4}$$
$$(a_2)^-1 \quad \bmod p \tag{5}$$
$$a_1 / a_2 \quad \bmod p \tag{6}$$

can be computed in $O(1)$ time on a $\lceil \log p \rceil \times p$ reconfigurable mesh.

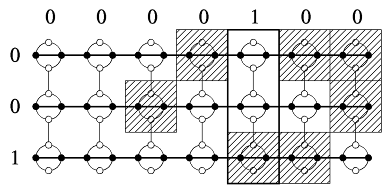


Fig. 3. A 3×7 reconfigurable mesh to multiply two integers in $[0, 6]$. The generator of $(\mathbb{Z}/7\mathbb{Z})^*$ chosen is three. Thus, $P_{i,j}$ is shaded if and only if the *i*th bit of $f(j)$ is equal to one, where $j \equiv 3^{f(j)} \bmod p$. In this example, the POS representation of five is input on the first row and the BIN representation of five is output (note that $3^5 \equiv 5 \bmod 7$).

Proof. Proof of (4): Let $c \in (\mathbb{Z}/p\mathbb{Z})^*$ be a generator of $(\mathbb{Z}/p\mathbb{Z})^*$ and let $f : [0, p-1] \to [0, p-2]$ be a function such that

$$j \equiv c^{f(j)} \mod p, \text{ for all } 0 < j < p.$$

Note that elementary algebra assures that, for each prime number p, there exists such an integer c and that f, restricted to the interval $[1, p-1]$, is a bijective function. (See Fig. 3.)

We can assume that $a_1, a_2 > 0$, indeed, the first row of the reconfigurable mesh can check whether one of the two integers is zero, in this case, the algorithm stops in $O(1)$ time giving zero as output.

By Lemma 3.3, letting $m = n$ and, by Corollary 3.4, the POS representations of d_1 and d_2 can be computed such that $c^{d_1} \equiv a_1 \mod p$ and $c^{d_2} \equiv a_2 \mod p$. Moreover,

$$a_1 \times a_2 \equiv c^{d_1} \times c^{d_2} \equiv c^{d_1 + d_2 \mod (p-1)} \mod p,$$

hence, $a_1 \times a_2$ can be computed in the following manner. By (1) of Lemma 3.5, $d_1 + d_2 \mod (p-1)$ is computed and, by Lemma 3.3 again, $c^{d_1+d_2 \mod (p-1)} \mod p$ is obtained, that is, $a_1 \times a_2 \mod p$.

Proof of (5): As in the proof of (4), d_2 can be found such that $c^{d_2} \equiv a_2 \mod p$. By (2) of Lemma 3.5, $-d_2 \mod (p-1)$ can be computed. Then, using the technique shown above, $c^{-d_2 \mod (p-1)} \mod p$, that is, $(a_2)^{-1} \mod p$ can be obtained.

Proof of (6): Note that $a_1/a_2 \equiv a_1 \times (a_2)^{-1} \mod p$. Hence, this result can be easily proven combining the proofs of (4) and (5). □

3.2 The Residue Number System Representations

In this section, the RPOS, R1UN, R2UN, and RBIN representations are introduced which are the RNS counterpart of the POS, 1UN, 2UN, and BIN representations considered in Section 3.1. In particular, algorithms for performing basic arithmetic operations using the RNS representations are also illustrated.

Definition 3.7. *Let $(m_1, \ldots, m_s)$ be an ordered set of positive integers all greater than one. Given an integer a, the RNS representation of a is an s-tuple $(a_1, \ldots, a_s)$, where a_i is the remainder of the division a/m_i, for all $i = 1, \ldots, s$. The m_is are called moduli, while the a_is are called remainders.*

By the Chinese Remainder Theorem we know that if the moduli are pairwise relatively prime, the RNS representation of two integers $a, b \in [0, \ldots, M-1]$, where M is the product of all the moduli, is identical if and only if $a = b$.

In the wide literature concerning Residue Number Systems, the problem of which representation has to be used for the remainders is often disregarded since the binary representation is implicitly accepted. It is important to note that, in this case, the choice of the moduli is not so crucial from a memory-usage point of view. Indeed, if the moduli are pairwise relatively prime, the area needed for storing the binary representations of the remainders for the largest integer $a \in [0, n-1]$ is the same as the area needed by the binary representation of a itself, within constant factors. This can be immediately deduced from

$$\log \prod_i m_i = \sum_i \log m_i. \tag{7}$$

This is not the case for reconfigurable meshes, where the unary representations are widely used. These remarks lead to the definition of the following representations.

Definition 3.8. *Let $p_1, \ldots, p_k$ be the k smallest prime numbers. We define the RPOS, R1UN, R2UN, and RBIN representations of an integer a as the RNS representation of a with moduli $p_1, \ldots, p_k$, where the remainders are represented using the POS, 1UN, 2UN, and BIN representations, respectively.*

Similarly to Theorem 1 of [15], the size of the above representations can be easily derived starting from the Prime Number Theorem. In particular, assume that we want to represent positive integers in $[0, n-1]$ and that M is equal to the product of the moduli $p_1, \ldots, p_k$. Then, in order to be able to uniquely represent all integers in $[0, n-1]$, it is easy to see that:

$$k = \Theta\left(\frac{\log n}{\log \log n}\right). \tag{8}$$

Consequently,

$$\log M = \Theta(\log n) \tag{9}$$

and p_k is asymptotically equal to $\log n$, within constant factors:

$$p_k = \Theta(\log n). \tag{10}$$

In the case of RPOS, R1UN, and R2UN, the representation $(r_1, \ldots, r_k)$ of an integer $a \in [0, n-1]$ is such that each remainder r_i needs p_i bits to be stored. Thus, the amount of bits needed by these representations is

$$\Theta\left(\frac{\log^2 n}{\log \log n}\right).$$

Therefore, these representations, where the remainders are stored in unary format, turn out to be quite compact.

In the case of RBIN, each remainder r_i needs $\lceil \log p_i \rceil$ bits, so the overall size taken by an integer in $[0, n-1]$ is

$$\Theta(\log n).$$

Note that the RBIN representation of an integer in $[0, M-1]$ needs as much area as the BIN representation of an integer in $[0, n-1]$, within constant factors.

Table 1 shows the growth of p_k, $\log_{10} M$, $\log_2 M$, and $\sum_{i=1}^{k} p_i$ as k increases. It can be clearly seen that the compactness of RPOS, R1UN, R2UN, and RBIN is not only

TABLE 1
Growth of Some Functions as k, the Number of Primes Used in the RPOS Representation, Increases

k	p_k	$\log_{10} M$	$\log_2 M$	$\sum p$
1	2	0.30	1.00	2
2	3	0.78	2.58	5
3	5	1.48	4.91	10
4	7	2.32	7.71	17
5	11	3.36	11.17	28
6	13	4.48	14.87	41
7	17	5.71	18.96	58
8	19	6.99	23.21	77
9	23	8.35	27.73	100
10	29	9.81	32.59	129
11	31	11.30	37.55	160
12	37	12.87	42.75	197
13	41	14.48	48.11	238
14	43	16.12	53.54	281
15	47	17.79	59.09	328
16	53	19.51	64.82	381

$M-1$ is the largest integer representable, p_k is the kth smallest prime, $\log_{10} M$, $\log_2 M$, and $\sum p$ are the digits needed to represent $M-1$ in the decimal, BIN, and RPOS representations.

asymptotically good, as shown above, but also quite satisfying for small numbers. This fact could be very important for applicative purposes. For example, to have the same power of representation of BIN with 32 bits, we need to use RPOS with moduli $p_1, \dots, p_{10}$, for a total amount of 129 bits.

In the remainder of the paper, p_i always denotes the ith smallest prime number, k the minimum number of primes needed to represent integers in $[0, n-1]$, and $M-1$ the biggest number representable with the RNS representation using $p_1, \dots, p_k$ as moduli.

We can easily extend Lemmas 3.5 and 3.6 to the new RPOS representation. Indeed, all the Residue Number System representations are well-suited for parallelization purposes since a wide range of operations can be done separately on each remainder.

Proposition 3.9. *Given the RPOS representation of two integers $a_1, a_2 \in [0, n-1]$ in a row, the RPOS representation of*

$$a_1 + a_2 \quad \bmod M \qquad (11)$$
$$-(a_2) \quad \bmod M \qquad (12)$$
$$a_1 - a_2 \quad \bmod M \qquad (13)$$
$$a_1 \times a_2 \quad \bmod M \qquad (14)$$

can be computed on an $O(\log\log n) \times O\left(\frac{\log^2 n}{\log\log n}\right)$ reconfigurable mesh in $O(1)$ time.

Proof. Thanks to the well-known properties of the RNS representations, all these operations can be done separately on each remainder. We use a $\log p_k \times \sum_{i=1}^{k} p_i$ reconfigurable mesh, built by serially attaching k $\log p_k \times p_i$ submeshes S_{p_i}, for $i = 1, \dots, k$ (see Fig. 4). The operations done within each submesh are identical to those described in Lemma 3.5 and Lemma 3.6. □

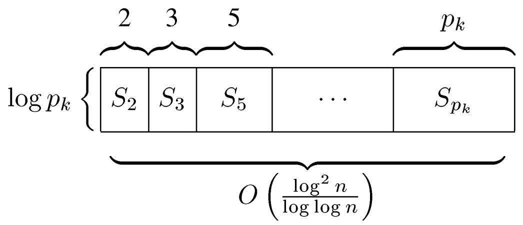


Fig. 4. The $\log p_k \times \sum_{i=1}^{k} p_i$ reconfigurable mesh used to perform the arithmetic operations described in Proposition 3.9.

3.3 Functions on Unary Notation

Unary notation has been widely used in reconfigurable meshes to obtain efficient constant-time algorithms.

We are interested in computing injective functions on positive integers in their POS representation. Let $A = \{a_1, \dots, a_h\}$ be a nonempty subset of $[0, \dots, n-1]$ and f an injective function from A to $[0, \dots, n-1]$. We can compute f on an $m \times n$ reconfigurable mesh building $|A|$ distinct linear subbuses, where subbus i has one endpoint at the N port of P_{0,a_i} and the other endpoint at the S port of $P_{m-1,f(a_i)}$. In this manner, if the n-bit POS representation of a number x is given at the upper boundary of the mesh one bit per processor, the POS representation of $f(x)$ will appear at the lower boundary in constant time. More formally, if $b_{n-1}b_{n-2}\cdots b_0$, the POS representation of $x \in A$ is given as input by putting a b_i-signal on the N port of $P_{0,i}$ for all i, then, after a constant time, the signals $c_{n-1}c_{n-2}\cdots c_0$ carried by the S ports of the last row of processors will represent the integer number $f(x)$ in the POS notation.

The sufficient and necessary conditions needed to build these subbuses on the $m \times n$ reconfigurable mesh can be derived from a paper by Frank [8]. His work concerns the disjoint paths problem on a rectilinear grid using a knock-knee model. It is fairly easy to see that his solution is directly portable to the linear reconfigurable mesh model and, so, the problem of building the subbuses is solved as a natural corollary of Frank's Theorem 1. It is worth noting that Frank's solution is algorithmic, i.e., it provides a procedure that effectively builds the paths and, hence, the switch configurations for the mesh.

Consider a vertical cut of the reconfigurable mesh. Surely the number of subbuses that must cross the cut is an important parameter to determine the number of rows needed for computing a function f.

Definition 3.10. *The congestion of f is the following number:*

$$c(f) = \max_j c_j(f),$$

where

$$c_j(f) = \#(\{i \in A \mid i \le j \text{ and } f(i) > j\} \cup \cup \{i \in A \mid i > j \text{ and } f(i) \le j\}).$$

It is evident that the congestion of a function is a lower bound on the number of rows of a reconfigurable mesh

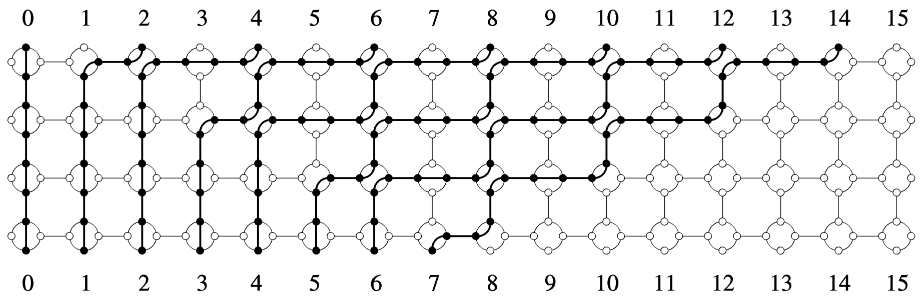


Fig. 5. Example of subbus system on a 4×16 reconfigurable mesh such that the port N of $P_{0,2x}$ is connected to the port S of $P_{3,x}$, for $0 \leq x < 8$.

capable of building the disjoint paths as defined above. However, it is quite surprising that, under proper hypotheses, it is an upper bound, too.

Definition 3.11. *We say that f satisfies the* column criterion *if and only if the number of rows of the mesh is greater than or equal to the congestion of f.*

Definition 3.12. *$P_{i,j}$ is called a* terminal vertex *if and only if either $i = 0$ and $j \in A$ or $i = m-1$ and $j \in f(A)$.*

We restate Frank's Theorem 1 in terms of reconfigurable subbuses.

Theorem 3.13 ([8]). *If at least one among $P_{0,0}$, $P_{0,n-1}$, $P_{m-1,0}$, and $P_{m-1,n}$ is not a terminal vertex, the column criterion is necessary and sufficient to build $|A|$ subbuses such that subbus i has one endpoint at the N port of P_{0,a_i} and the other at the S port of $P_{m-1,f(a_i)}$.*

Several results present in the literature can be viewed as simple corollaries of Theorem 3.13. Followed is an example, where, for the sake of simplicity, n is supposed to be even. This result is a slight improvement on Lemma 2 in [10], where an $\lfloor \frac{n}{2} \rfloor \times n$ reconfigurable mesh is used. Although Corollary 3.14 improves only by a factor of two on Lemma 2 in [10], it represents a stronger result. Indeed, according to the above discussion, $\lfloor \frac{n}{4} \rfloor$ is exactly the minimum number of rows needed to build the subbus system to perform $\lfloor x/2 \rfloor$ in one broadcasting step and it is remarkable that this derives as a simple consequence of Theorem 3.13.

Corollary 3.14. *Given in a row the n-bit POS representation of an integer x, an $\lfloor \frac{n}{4} \rfloor \times n$ reconfigurable mesh can compute $\lfloor x/2 \rfloor$ and $x \bmod 2$ in constant time.*

Proof. The proof can be derived from Theorem 3.13 by letting $A = \{0, 2, 4, \ldots, n-2\}$ and $f(x) = \frac{x}{2}$. f is clearly injective and $P_{m-1,n-1}$ is not a terminal vertex, so proving that the congestion of f is less than or equal to $\lfloor \frac{n}{4} \rfloor$ is sufficient to obtain the thesis by Theorem 3.13 (see Fig. 5 as an example).

$$\begin{aligned} c_j(f) &= \#(\{i \in A \mid i \leq j \text{ and } f(i) > j\} \cup \\ &\quad \cup \{i \in A \mid i > j \text{ and } f(i) \leq j\}) = \\ &= \{i \in A \mid i > j \text{ and } f(i) \leq j\} = \\ &= \min\left\{ \left\lceil \frac{j}{2} \right\rceil, \left\lfloor \frac{n-j}{2} \right\rfloor \right\} \leq \left\lfloor \frac{n}{4} \right\rfloor. \end{aligned}$$

Without loss of generality, we can suppose that x is stored in the first row of the mesh. Hence, $P_{0,x}$ stores 1, while $P_{0,i}$ stores 0, for all $i \neq x$. During the first step of the computation, $P_{0,x}$ transmits a signal to $P_{0,x-1}$ (if this processor exists). Then, both the processors $P_{0,x}$ and $P_{0,x-1}$ broadcast a 1-signal on their N port. Obviously, one and only one of the two integers x and $x-1$ is even and, consequently, only $P_{m-1,\lfloor x/2 \rfloor}$ receives a 1-signal. Hence, the last row of the mesh receives the POS representation of $\lfloor x/2 \rfloor$. Note that if $x = 0$, the algorithm still works, even if $P_{0,x-1}$ does not exist.

$x \bmod 2$ can be easily computed by transmitting a signal from the N port of $P_{0,x}$ and performing a NOR operation on the signals received by the last row of the mesh. □

It is useful to state another simple corollary of Theorem 3.13. It assures that n rows and $n+1$ columns are enough to build the subbuses according to any injective function f.

Corollary 3.15. *Let $f : [0, n-1] \rightarrow [0, n-1]$ be an injective function. It is possible to build n subbuses on an $n \times (n+1)$ reconfigurable mesh such that subbus i has one endpoint at the N port of $P_{0,i}$ and the other endpoint at the S port of $P_{n-1,f(i)}$.*

Proof. Noting that the congestion of f is less than or equal to n and that $P_{0,n}$ is not a terminal vertex, we can get the thesis by Theorem 3.13. □

3.4 Conversions and Prefix Sums of N Bits

This section illustrates the techniques for all the possible conversions between pairs of representations. In particular, the BIN $\rightarrow$ RPOS $\rightarrow$ BIN conversions are the most important since the BIN representation is usually employed for input and output data, while the RPOS representation introduced in Section 3.2 is well suited for efficient parallel processing on a reconfigurable mesh. The BIN $\rightarrow$ RPOS conversion follows by Lemma 3.23 and Proposition 3.24, while the RPOS $\rightarrow$ BIN conversion derives from Corollary 3.16 and Proposition 3.26. As a byproduct of the conversion techniques, an efficient algorithm for computing the Prefix Sums of N bits is given by Theorem 3.19 and Corollary 3.20.

Corollary 3.16. *Given an integer $a \in [0, n-1]$ in its RPOS representation in a column, its RBIN representation can be computed in $O(1)$ time on an $O(\frac{\log^2 n}{\log \log n}) \times O(\log n)$ reconfigurable mesh and vice versa.*

Proof. The conversion algorithm is the same as in Corollary 3.4 applied in parallel on each remainder of

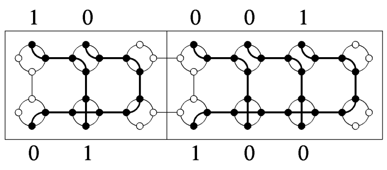


Fig. 6. A subbus system capable of adding 1 to the RPOS representation of an integer given on the first row. In this example, k is equal to two, the RPOS representation of two is given on the first row, and the RPOS representation of three is obtained on the last row.

the RPOS representation. To perform this operation an $O(\frac{\log^2 n}{\log\log n}) \times O(\log\log n)$ reconfigurable mesh is sufficient, but if we want to compact the RBIN representation of a using a single row, we need a larger mesh. This task can be easily executed by standard data moving techniques.□

Proposition 3.17. *Given the POS representation of an integer $a \in [0, n-1]$ in a row, a can be converted into its RBIN representation in $O(1)$ time on an $O(\log n) \times n$ reconfigurable mesh and vice versa.*

Proof. The $O(\log n) \times n$ reconfigurable mesh is arranged in k horizontal slices S_i, $1 \leq i \leq k$. Each S_i is composed of $\lceil \log p_i \rceil$ rows. By using column subbuses, the POS representation of a can be broadcast to all the rows of the mesh.

Let $f_i : [0, n-1] \rightarrow [0, p_i - 1]$ be such that $f_i(a) \equiv a \bmod p_i$. By Lemma 3.3, letting $m = p_i$, each submesh S_i can compute the BIN representation of the remainder of a modulo p_i. Thus, the RBIN representation of a can be stored in a column.

Now, suppose that the RBIN representation of a is stored in a column. By Lemma 3.3, each S_i can compute the set A_i of integers $j \in [0, n-1]$ such that $a \equiv j \bmod p_i$. By the Chinese Remainder Theorem, only $a \in [0, n-1]$ is in the set A_i, for all $1 \leq i \leq k$. Thus, a can be found in constant time by performing in parallel an AND operation along each column of the mesh. □

Corollary 3.18. *Given the POS representation of an integer $a \in [0, n-1]$ in a row, a can be converted into its RPOS representation in $O(1)$ time on a $\log n \times n$ reconfigurable mesh and vice versa.*

Proof. The proof follows by Proposition 3.17 and Corollary 3.16. □

One of the most common operations for applicative purposes is the sum of N bits. This problem has already been solved using a reconfigurable model, see Corollary 1 in [14], where an $N \times N$ reconfigurable mesh is used to compute all the prefix sums of N bits in $O(1)$ time. An improvement on this result was given in [15], where an $N \times \frac{\log^2 N}{\log\log N}$ common-write word model reconfigurable mesh is used. By following the same approach of [14] and [15], the same result can be obtained on an exclusive write bit model reconfigurable mesh of the same size, within constant factors, as a simple application of the RNS number representations defined in this paper.

Theorem 3.19. *Given N bits, the prefix sums of these bits can be computed in $O(1)$ time on a $2N \times O(\frac{\log^2 N}{\log\log N})$ reconfigurable mesh.*

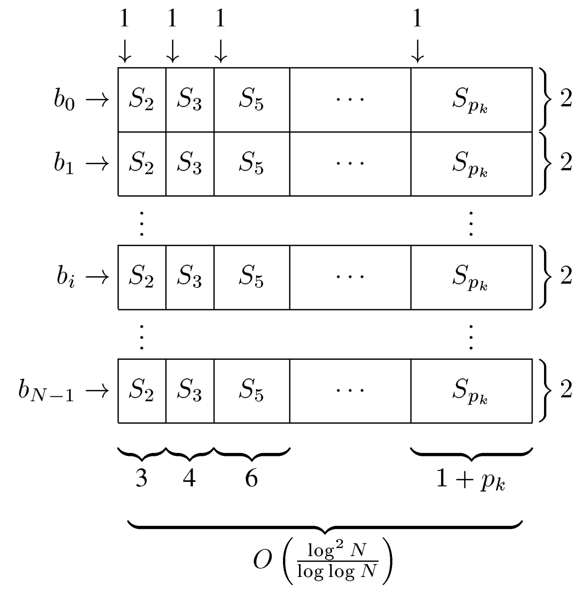


Fig. 7. A scheme of reconfigurable mesh capable of summing N bits in $O(1)$ time.

Proof. Clearly, the sum of N bits is an integer in $[0, \ldots, N]$. Thus, we will use the smallest RPOS representation capable to represent N.

Let $f_i : [0, \ldots, p_i] \rightarrow [0, \ldots, p_i]$, $i = 1, \ldots, k$, be k functions defined as follows:

$$f_i(x) = (x + 1) \bmod p_i;$$

for each i, f_i is injective and its congestion is two. Thus, according to Theorem 3.13, using a $2 \times (p_i + 1)$ reconfigurable mesh S_{p_i} it is possible to build p_i subbuses such that subbus x has one endpoint at the N port of $P_{0,x}$ and the other endpoint at the S port of $P_{1,f_i(x)}$. Serially attaching the k reconfigurable meshes $S_{p_1}, S_{p_2}, \ldots S_{p_k}$ (as shown in Fig. 6), we get a $2 \times O(\frac{\log^2 N}{\log\log N})$ mesh S capable of summing one to the RPOS representation of an integer given on the upper boundary. A larger $2N \times O(\frac{\log^2 N}{\log\log N})$ reconfigurable mesh H (that solves the problem) can be obtained stacking together N identical copies of the submesh S (see Fig. 7). Each of these N submeshes gets as input a bit b_i from the left. This bit has to be broadcast to all the PEs of the submesh S. According to the value of the bit, S either configures its switches to perform a +1 operation, if the bit is 1, or builds column subbuses to perform a +0 operation, if the bit is 0. Putting on the upper boundary of H the RPOS representation of zero, we get, in $O(1)$ time, the RPOS representation of all the prefix sums of the n bits from the lower boundary of the ith copy of S, $1 \leq i \leq N$. Thus, the overall sum of the n bits is gotten from the lower boundary of H itself. To get the POS representation of this value, we can use Corollary 3.18 on H by swapping columns and rows. In an analogous way we can get its BIN representation using Proposition 3.4. □

The above results can be used to derive a simple summation algorithm, useful to perform some conversions between pairs of representations. The proof follows a technique used earlier in [11].

Corollary 3.20. *Given N bits, the sum of these bits can be computed in $O(1)$ time on an $N \times O(\frac{\log^2 N}{\log\log N})$ reconfigurable mesh.*

Proof. The reconfigurable mesh H shown in the proof of the previous theorem can be modified by halving the number of rows. The resulting $N \times O(\frac{\log^2 N}{\log\log N})$ mesh H' can iterate twice the algorithm of H on a couple of bits b_i and b_{i+1} per submesh S got as input from the left. H' sums the bits in even position during the first step, broadcasts the partial sum from the last row to the first one, and configures itself to sum the bits in odd position during the second step. In this way, H' is capable of summing N bits in $O(1)$ time, as claimed.

As in the proof of Theorem 3.19, we can easily get the POS and BIN representations of the computed sum. □

The above Summation algorithm is used for performing some conversions between pair of representations, as shown in the following two corollaries.

Corollary 3.21. *Given the 2UN representation of an integer $a \in [0, n-1]$ in a column, a can be converted into its POS representation in $O(1)$ time on an $n \times O(\frac{\log^2 n}{\log\log n})$ reconfigurable mesh.*

Proof. The conversion simply consists of adding the n bits of the 2UN representation on an $n \times O(\frac{\log^2 n}{\log\log n})$ reconfigurable mesh, as shown in Corollary 3.20, letting $N = n$. □

Corollary 3.22. *Given the R2UN representation of an integer $a \in [0, n-1]$ in a column, its RPOS representation can be computed in $O(1)$ time on an $O(\frac{\log^2 n}{\log\log n}) \times O(\frac{\log^2 \log n}{\log\log\log n})$ reconfigurable mesh.*

Proof. Consider k submeshes M_i, $1 \le i \le k$, obtained with the rows of the mesh storing the ith remainder of a. The size of M_i is $p_i \times O(\frac{\log^2 \log n}{\log\log\log n})$.

By letting $N = p_i$ in Corollary 3.20, each M_i can convert in parallel the remainder it stores into its POS representation by adding the bits of the 2UN representation. In this way, the RPOS representation of a is computed in $O(1)$ time, as required. □

Before showing how to perform the BIN $\rightarrow$ RBIN and BIN $\rightarrow$ RPOS conversions, the following preliminary lemma is needed.

Lemma 3.23. *Given the BIN representation of an integer $a \in [0, n-1]$ in a row, the BIN (POS) representation of $a \bmod p_i$ can be computed in $O(1)$ time on an $O(\log n) \times O(\log n)$ reconfigurable mesh.*

Proof. Let $h = \lceil \log n \rceil$ and let $(a_0, a_1, \dots, a_{h-1})$ be the BIN representation of a. We can partition the h bits of the representation of a into groups of $w = \max\{\lceil \log\log n \rceil, \lceil \log p_i \rceil\}$ bits each. Thus, letting $a_\beta = 0$, for all $\beta \ge h$,

$$a \equiv \left(\sum_{s=0}^{h-1} a_s 2^s \right) \equiv$$
$$\equiv \left(\sum_{j=0}^{\lceil h/w \rceil - 1} 2^{jw} \sum_{s=0}^{w-1} a_{jw+s} 2^s \right) \bmod p_i.$$

Let

$$b_j = \sum_{s=0}^{w-1} a_{jw+s} 2^s, \quad 0 \le j < \lceil h/w \rceil,$$

then,

$$a \equiv \left(\sum_{j=0}^{\lceil h/w \rceil - 1} \left(b_j 2^{jw} \bmod p_i \right) \right) \bmod p_i.$$

Suppose that the reconfigurable mesh is arranged in $\lceil h/w \rceil$ vertical slices S_j, $0 \le j < \lceil h/w \rceil$, such that S_j has the BIN representation of b_j in a row. By Corollary 3.4, each S_j can get the POS representation of b_j in a column.

Let $f_j : [0, 2^w - 1] \rightarrow [0, \lceil \log p_i \rceil - 1]$ be a bijective function such that $f_j(x) = x 2^{jw} \bmod p_i$. By Lemma 3.3, each S_j can compute $f_j(b_j)$ and store it in a row. Therefore,

$$a \bmod p_i = \left(\sum_{j=0}^{\lceil h/w \rceil - 1} f_j(b_j) \right) \bmod p_i$$

and it is possible to use Lemma 2 of [9] since $\lceil h/w \rceil$ is $O(\frac{\log n}{\log\log n})$ and each $f_s(b_s)$, $0 \le s < \lceil h/w \rceil$, consists of $\log p_i = O(\log\log n)$ bits. Thus, by Lemma 2 of [9], an $O(\frac{\log n}{\log\log n}) \times O(\log n)$ reconfigurable mesh is sufficient to perform this sum. It is important to note that Lemma 2 needs the inputs in a particular permutation of the bits. So, the $O(\log n)$ bits of the numbers to be added must be routed to the correct places. This is surely possible on our reconfigurable mesh by well-known routing techniques since it has $O(\log n)$ rows and $O(\log n)$ columns.

Finally, the POS representation of the remainder, which needs $O(\log n)$ bits to be stored, can be obtained by Corollary 3.4. □

It is worth noting that either Theorem 1 of [9] could be used to improve the previous lemma or the lemma itself can be directly derived from Theorem 1 of [11]. However, such strong results are not needed in the remainder of this paper. Moreover, the algorithm of Lemma 3.23 seems to be more suited for applicative purposes since it requires less broadcasting operations. The following proposition gives an algorithm for performing the BIN $\rightarrow$ RBIN and BIN $\rightarrow$ RPOS conversions.

Proposition 3.24. *Given the BIN representation of an integer $a \in [0, n-1]$ in a row, the RBIN (RPOS) representation of a can be computed in $O(1)$ time on an $O\left(\frac{\log^2 n}{\log\log n}\right) \times O(\log n)$ reconfigurable mesh.*

Proof. We use a reconfigurable mesh obtained by stacking together k $O(\log n) \times O(\log n)$ submeshes. The ith submesh, $1 \le i \le k$, is charged of computing the ith

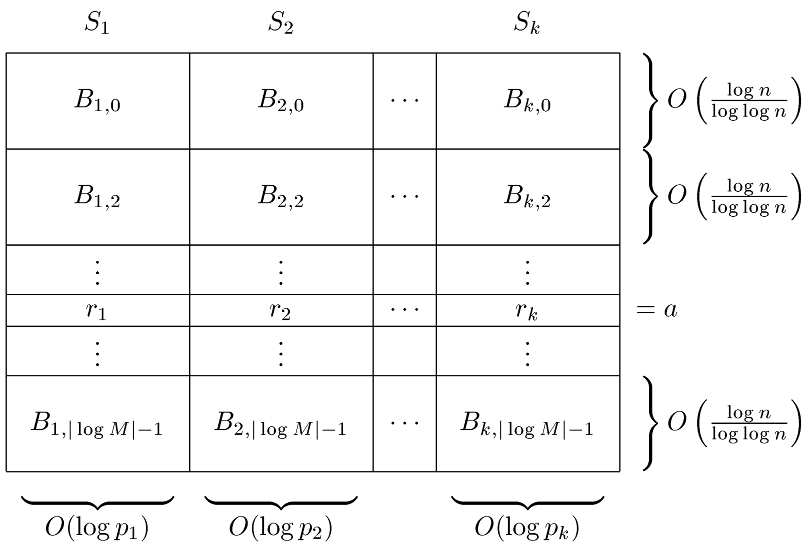

S_1
S_2
S_k
$B_{1,0}$
$B_{2,0}$
$\cdots$
$B_{k,0}$
$O\left(\frac{\log n}{\log\log n}\right)$
$B_{1,2}$
$B_{2,2}$
$B_{k,2}$
$O\left(\frac{\log n}{\log\log n}\right)$
r_1
r_2
r_k
$= a$
$B_{1,\lceil\log M\rceil-1}$
$B_{2,\lceil\log M\rceil-1}$
$B_{k,\lceil\log M\rceil-1}$
$O\left(\frac{\log n}{\log\log n}\right)$
$O(\log p_1)$
$O(\log p_2)$
$O(\log p_k)$

Fig. 8. Structure of the reconfigurable mesh used in the proof of Proposition 3.26.

remainder of the RBIN (RPOS) representation of a by Lemma 3.23. Since $k = O\left(\frac{\log n}{\log\log n}\right)$, the size of the mesh is as stated. □

In order to show the RBIN → BIN and RPOS → BIN conversions in the next Proposition 3.26 and Corollary 3.27, respectively, the following preliminary result will be useful.

Lemma 3.25. *Let $f : [0, m-1] \to [0,1]$ be a function. Given the BIN representation of $a \in [0, m-1]$ in a row, an $O(m/\log m) \times \lceil \log m \rceil$ reconfigurable mesh can compute $f(a)$ in constant time.*

Proof. Suppose that processor $P_{i,j}$ stores a bit equal to $f(j2^{i\lfloor\log\lfloor\log m\rfloor\rfloor})$.

We can arrange the reconfigurable mesh into two submeshes S_1 and S_2 serially attached. S_1 has $O(m/\log m)$ rows and $\lceil \log m\rceil - \lfloor\log\lfloor\log m\rfloor\rfloor$ columns, S_2 has $O(m/\log m)$ rows and $\lfloor\log\lfloor\log m\rfloor\rfloor$ columns. Thus, both S_1 and S_2 can separately compute the POS representation of the bits of a stored in their row (note that $\lfloor\log m\rfloor = O(m/\log m)$ rows are needed to perform this operation). The POS representation stored in S_2, consisting of at most $\lfloor \log m\rfloor$ bits, can be broadcast to all the rows of the mesh. Similarly, the POS representation stored in S_1, consisting of $O(m/\log m)$ bits, can be broadcast to all the columns. In this way, the only processor $P_{i,j}$ which stores the one of both representations is such that $a = j2^{i\lfloor\log\lfloor\log m\rfloor\rfloor}$. Thus, it can output the value $f(j2^{i\lfloor\log\lfloor\log m\rfloor\rfloor}) = f(a)$. □

Proposition 3.26. *Given the RBIN representation of an integer $a \in [0, n-1]$ in a row, its BIN representation can be computed in $O(1)$ time on an $O(\frac{\log^2 n}{\log\log n}) \times O(\log n)$ reconfigurable mesh.*

Proof. Let $(r_1, \ldots, r_k)$ be the RBIN representation of a. It is well-known that the BIN representation of a can be computed in the following manner. Let

$$M_i = \frac{M}{p_i}, \quad 1 \le i \le k,$$

and define T_i to be the inverse of M_i modulo p_i, that is

$$T_i M_i \equiv 1 \bmod p_i, \quad 1 \le i \le k.$$

Then,

$$a = \sum_{i=1}^{k} r_i T_i M_i \bmod M. \tag{15}$$

The BIN representation of T_iM_i uses $O(\log n)$ bits.

We use a reconfigurable mesh arranged in k submeshes S_i, $i = 1, \ldots, k$, serially attached. Each submesh S_i has $O(\frac{\log^2 n}{\log\log n})$ rows and $O(\log p_i)$ columns in such a way that submesh S_i stores the ith remainder of a in a row. Moreover, each S_i is arranged in $O(\log n)$ submeshes B_{ij}, $j = 0, \ldots, \lceil\log M\rceil - 1$, each of size $O(\frac{\log n}{\log\log n}) \times O(\log p_i)$ (See Fig. 8).

The algorithm acts as follows: During the first step, each S_i broadcasts the ith remainder to all the B_{ij} submeshes, $j = 0, \ldots, \lceil \log M\rceil - 1$. Then, by Lemma 3.25, each B_{ij}, for all $i = 1, \ldots, k$ and $j = 0, \ldots, \lceil\log M\rceil - 1$, computes in constant time the value $f_{ij}(r_j)$, where $f_{ij} : [0, p_i - 1] \to [0,1]$ and $f_{ij}(r)$ is equal to the jth bit of the binary representation of rM_iT_i.

Now, all the jth bits of the binary representation of the addenda of (15) are stored in a horizontal slice, consisting of all the B_{ij}, $i = 1, \ldots, k$, of the reconfigurable mesh. Thus, it is possible to compute the binary representation of

$$s = \sum_{i=1}^{k} r_i T_i M_i \tag{16}$$

by Lemma 2 of [9]. Note that $s \le kM$ since s is the sum of k numbers each smaller than M. Therefore, the modulo M operation can be easily performed using a look-up table of size $k \times \lceil\log M\rceil$, storing iM, $i = 1, \ldots, k$. The largest d such that dM is smaller than or equal to s can be easily found and then a is obtained by performing $s - dM$. □

Corollary 3.27. *Given the RPOS representation of an integer $a \in [0, n-1]$ in a column, its BIN representation can be computed in $O(1)$ time on an $O(\frac{\log^2 n}{\log\log n}) \times O(\log n)$ reconfigurable mesh.*

Proof. The proof follows by Corollary 3.16 and Proposition 3.26. □

All the results shown in this paper about conversions between pairs of representations can be summarized in the following main result. Corollary 3.28 outperforms Lemma 1 of [10], which uses an $N \times N$ word model of reconfigurable mesh and is limited to the representations defined in Section 3.1.

Corollary 3.28. *Given an $n \times O\left(\frac{\log^2 n}{\log\log n}\right)$ reconfigurable mesh, an integer $a \in [0, n-1]$ can be converted in constant time from any representation defined in this paper to any other representation.*

4 Adding N h-Bits Numbers

This section presents new algorithms for computing the Summation and the Prefix Sums of N h-bit numbers. Given N integers $y_0, \ldots, y_{N-1}$, the Prefix Sums problem consists of computing the integers $z_0, \ldots, z_{N-1}$ such that $z_i = y_0 + \cdots + y_i$, $i = 0, \ldots, N-1$. When i is fixed to $N-1$ only, the Summation problem results. The next theorem, 4.1, shows how to compute the Summation in $O(1)$ time using an $O\left(h\log N + \frac{\log^2 N}{\log\log N}\right) \times Nh$ reconfigurable mesh.

Theorem 4.1. *Given the BIN representations of N h-bit numbers, $1 \leq h \leq N$, such that the jth bit of the ith number is stored in $P_{0,i+jN}$, for all $0 \leq i < N$ and $0 \leq j < h$, these numbers can be added in $O(1)$ time on a reconfigurable mesh of size $O\left(h\log N + \frac{\log^2 N}{\log\log N}\right) \times Nh$.*

Proof. The reconfigurable mesh is arranged in h submeshes S_j, $0 \leq j < h$, each of size $O\left(h\log N + \frac{\log^2 N}{\log\log N}\right) \times N$, serially attached. Each S_j stores the jth bits of the N numbers in its first row. The algorithm acts as follows: By Corollary 3.20, each submesh S_i computes s_j, the number of bits set to one stored in its first row. The values s_j, $0 \leq j < h$, are such that

$$\sum_{w=0}^{h-1} s_w 2^w$$

is the overall sum that must be computed. To perform this sum, Lemma 2 of [9] can be used. Indeed, the $h \times h(h + \log N)$ reconfigurable mesh needed can be easily embedded in the mesh used in this proof. It is important to note that a proper permutation of the bits must be arranged. However, this permutation can be easily performed by standard techniques since the overall amount of data to be routed is $h\log N$, that is, less than or equal to the number of rows of the mesh. □

It is worth noting that Corollary 3.20 can be obtained from Theorem 4.1 by letting $h = 1$.

We now turn our attention to the design of an efficient algorithm for the Prefix Sums problem. The properties of RNS representations allow good parallelizations of many algorithms to be obtained as follows: First, convert the input given in BIN representation into an RNS representation (RPOS, for example), then perform the operations of the algorithm in parallel on each remainder, and finally, convert again the output into its BIN representation. Section 3 provided all the techniques needed to use such a kind of parallelization. This is the underlying idea of the algorithm to be presented in Theorem 4.3 for computing the Prefix Sums on an $O\left(\frac{h^2+\log^2 N}{\log(h+\log N)}\right) \times O(N(h + \log N))$ reconfigurable mesh.

Lemma 4.2. *Given the POS representations of N numbers in $[0, \ldots, x-1]$, the prefix sums modulo x of these numbers can be computed in $O(1)$ time on an $(1 + x) \times 2Nx$ reconfigurable mesh.*

Proof. The ith number is input one bit per processor to $P_{0,2ix}, P_{0,2(ix+1)}, \ldots, P_{0,2(ix+x-1)}$. The first row of the mesh can convert the N numbers in parallel to their 1UN representation, as stated in Lemma 3.2 (with minor changes due to the fact that the PEs in odd position must be ignored). Consider a reconfigurable mesh arranged into Nx submeshes B_i serially attached, $0 \leq i < Nx$, each of $(1 + x)$ rows and two columns. More precisely, B_i consists of the columns $2i$ and $2i + 1$ of the mesh. Then, we can execute a two-step algorithm. During the first step, the bits in even position are added. Each B_i broadcasts the bit in even position that it stores to all the PEs of the submesh. Then, it configures its switches in such a way that the W port of the jth processor in the first column is connected to the E port of the $[(j+1) \bmod x]$th processor in the second column, for $0 \leq j < x$. Using a well-known technique, if $P_{0,0}$ sends a 1-signal along its W port, the POS representation of the sum of the bits up to the 2ith column will appear on the right boundary of each B_i. Therefore, for all $0 \leq j < N$, the sum of the first j numbers is obtained in $O(1)$ time on the left boundary of B_{2jx}, as required. □

Theorem 4.3. *Given the BIN representation of N h-bit numbers in a row, the prefix sums of these numbers can be computed in $O(1)$ time on an $O\left(\frac{h^2+\log^2 N}{\log(h+\log N)}\right) \times O(N(h + \log N))$ reconfigurable mesh.*

Proof. Consider a reconfigurable mesh as shown in Fig. 9 Each of the S_i, $0 \leq i < N$, is capable of getting in input, on the first row, an integer y_i in its BIN representation. Each y_i is an h-bit number, but the first row of each S_i can easily expand it with zeros to have an $(h + \log N)$-bit number. By Proposition 3.24, the y_is can be converted in parallel to their RPOS representations using an $O(\frac{h^2+\log^2 N}{\log(h+\log N)}) \times O(h + \log N)$ reconfigurable mesh. Thus, the size of each S_i is $O(\frac{h^2+\log^2 N}{\log(h+\log N)}) \times O(h + \log N)$.

Consider a partition of the mesh into k "horizontal" slices B_j, $1 \leq i \leq k$. Each B_j consists of the rows which contain all the jth remainders of the division of $y_0, \ldots, y_{N-1}$ by p_j.

B_j has $O(h + logN)$ rows and $O(N(h + \log N))$ columns. Thus, the algorithm of Lemma 4.2 can be performed to get, in each B_j, the prefix sums modulo

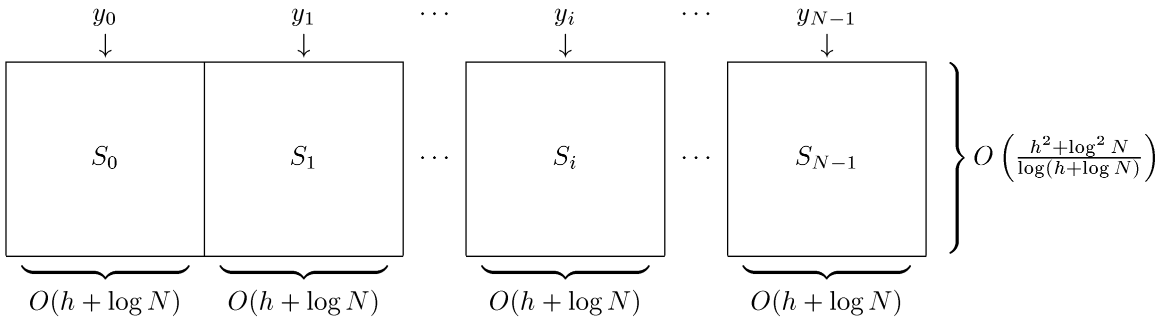


Fig. 9. Scheme of the $O\left(\frac{h^2+\log^2 N}{\log(h+\log N)}\right) \times O(N(h+\log N))$ reconfigurable mesh used in the proof of Theorem 4.3. Each S_i is similar to the mesh described in Proposition 3.24 and has $O\left(\frac{h^2+\log^2 N}{\log(h+\log N)}\right)$ rows and $O(h+\log N)$ columns.

p_j of the jth remainders of the RPOS representation of $y_0, \ldots, y_{N-1}$. At this point, each S_i has the RPOS representation of the sum of $y_0, \ldots, y_i$, $0 \leq i < N$, and, by Corollary 3.27, can get the BIN representation of such a sum. □

To our knowledge, no constant time algorithm was devised before for computing the Prefix Sums on the bit model, the only constant time being that proposed in [18], which uses an $N \times N$ word model of reconfigurable mesh and works only when $h = O(\log N)$. Thus, the constant time Prefix Sums algorithm proposed in Theorem 4.3 is the first such algorithm for the bit model.

5 Approximate String Matching

This section presents another application of the parallelization techniques introduced in Section 3. Besides the Summation and Prefix Sums problems considered in Section 4, the parallelization techniques based on RNS representations are applied, in this section, to the Approximate String Matching problem with α mismatches. String Matching is one of the most important computational problems, both from a theoretical and an applicative viewpoint. Given a "pattern" and a "text," which are strings over a finite alphabet, one wants to find all the occurrences of the pattern within the text. Formally, let Σ be a finite alphabet and let $\Pi = \pi_0\pi_1\cdots\pi_{m-1}$ and $\mathrm{T} = \tau_0\tau_1\cdots\tau_{n-1}$ denote the pattern and the text, respectively, where $m \leq n$.

Definition 5.1. *The* String Matching *problem consists of finding all the occurrences of* Π *in* T. *Formally, we are interested in computing the function*

$$\phi : \Sigma^m \times \Sigma^n \rightarrow \{0,1\}^{n-m+1},$$

where

$$\phi(\Pi, \mathrm{T}) = b_0 b_1 \cdots b_{n-m},$$

and, for $0 \leq i \leq n-m$,

$$b_i = \begin{cases} 1 & \text{if } \pi_0 = \tau_i \text{ and } \ldots \text{ and } \pi_{m-1} = \tau_{i+m-1}, \\ 0 & \text{otherwise.} \end{cases}$$

Several variants of the exact String Matching problem defined above have been devised. Some of them turn out to be useful in biological applications, where solving pattern matching problems with very long texts is needed. In these cases, allowing a limited number of mismatches is often very important to find approximate occurrences of the pattern in the text.

Definition 5.2. *Let* α *be a positive integer less than* m. *The* Approximate String Matching with α Mismatches *problem consists of finding all the occurrences of* Π *in* T, *allowing a number of mismatches less than* α. *Thus, a function*

$$\psi : \Sigma^m \times \Sigma^n \times [1, \ldots, m] \rightarrow \{0,1\}^{n-m+1}$$

has to be computed, where

$$\psi(\Pi, \mathrm{T}, \alpha) = c_0 c_1 \cdots c_{n-m},$$

and, for $0 \leq i \leq n-m$,

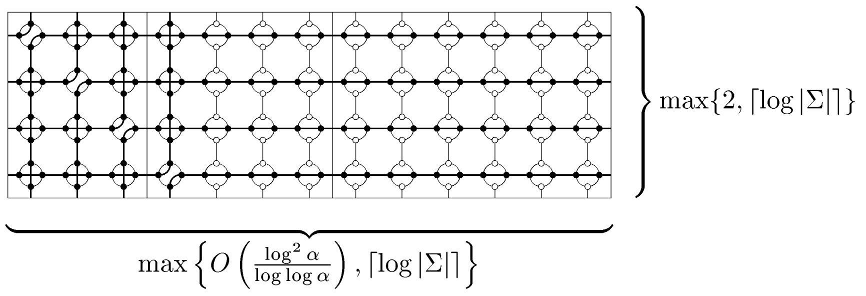
$$c_i = \begin{cases} 1 & \text{if } \#\{j \in [0, m-1] \mid \pi_j \neq \tau_{i+j}\} < \alpha, \\ 0 & \text{otherwise.} \end{cases}$$

Letting $\alpha = 1$ in Definition 5.2 clearly leads to the exact String Matching problem stated in Definition 5.1.

The exact String Matching problem has been considered and solved in $O(1)$ time using a reconfigurable mesh of size $m\lceil\log|\Sigma|\rceil \times n\lceil\log|\Sigma|\rceil$ [7]. To our knowledge, the Approximate String Matching with α mismatches problem has not been considered before using the reconfigurable mesh. The solution to be proposed here is a good example of nontrivial application of the techniques presented in Section 3.

Theorem 5.3. *Let* Π *and* α *be given on the first column and* T *on the first row, of a reconfigurable mesh of size* $O(m\log|\Sigma|) \times O\left(n\left(\log|\Sigma| + \frac{\log^2\alpha}{\log\log\alpha}\right)\right)$. *Then,* $\psi(\Pi, \mathrm{T}, \alpha)$ *can be computed in constant time.*

Proof. The reconfigurable mesh used to compute ψ is arranged as shown in Fig. 11. Each B_{ij}, $0 \leq i < m$, and $0 \leq j < n$, has $\max\{2, \lceil\log|\Sigma|\rceil\}$ rows and $\max\left\{O\left(\frac{\log^2\alpha}{\log\log\alpha}\right), \lceil\log|\Sigma|\rceil\right\}$ columns (see Fig. 10). Thus, it is capable of storing a character of Σ in a column and both a character and the RPOS representation of an integer belonging to $[0, \alpha]$ in a row. For each j, a submesh S_j, obtained combining B_{ij} for all $0 \leq i < m$, is defined. Without loss of generality, the integer α can be supposed to be input in its POS representation (this is possible



$\max\{2, \lceil \log |\Sigma| \rceil\}$
$\max\left\{O\left(\frac{\log^2 \alpha}{\log\log\alpha}\right), \lceil \log |\Sigma| \rceil\right\}$

Fig. 10. Structure of each single B_{ij} submesh, $0 \le i < m$ and $0 \le j < n$. In this example, $|\Sigma|$ is 4 and α is 29.

since $\alpha < m$). Indeed, Corollary 3.28 can be used to get the POS representation of α in constant time from whatever representation defined in Section 3.

With a broadcast operation, using row buses, the POS representation of α can be stored on the first column of each S_j. Then, each submesh S_j performs the algorithm of Corollary 3.18 to get the RPOS representation of α in its first row. This value can be broadcast in constant time to the first row of each B_{ij} within S_j. Another broadcast operation, using row buses, is needed to let each B_{ij} store π_i on its first column. Then, each B_{ij} configures its switches as shown in Fig. 10. In this way, subbuses along the antidiagonals of the reconfigurable mesh are built which can be used in parallel by each B_{0j}, for all $0 \le j < n$. Indeed, B_{0j} can broadcast the character τ_j it stores to the first $\lceil \log |\Sigma| \rceil$ processors of the submeshes $B_{h,j-h}$, for all $1 \le h < m$ such that $j - h \ge 0$. After these steps, each submesh B_{ij} stores p_i in the first column and α and τ_{i+j} on the first row of processors. With standard techniques, it is now easy to get B_{ij} to store either a one, if $\pi_i \ne \tau_{i+j}$, or a zero, otherwise.

Thus, each submesh S_j has all the information needed to compute c_j. S_j can perform in parallel, for all $0 \le j < n$, the algorithm described in Theorem 3.19 to get in each B_{ij}, $0 \le i < m$, the number of mismatches detected comparing the first i characters of the string Π with the substring of T starting from position j. This value is in RPOS representation and it can be easily compared with α to check whether they are equal. If the answer is "yes" and, thus, the number of errors is equal to α, B_{ij} stores 1, otherwise it stores 0. Finally, we can easily get c_j as the NOR of the m Boolean values computed in S_j.

All the steps described can clearly be performed in constant time on the reconfigurable mesh, so the whole algorithm requires constant time, too. $\square$

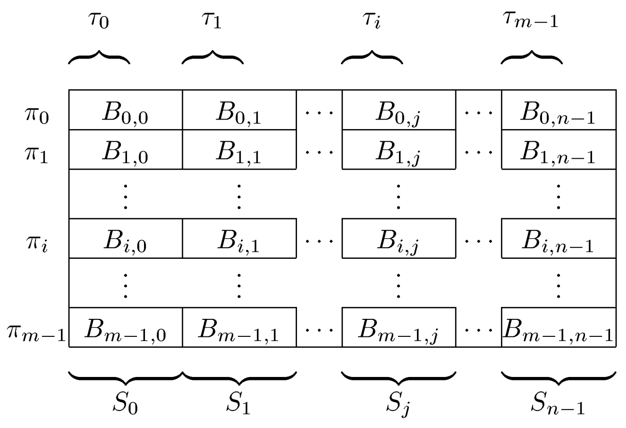


Fig. 11. Scheme of the reconfigurable mesh used in the proof of Theorem 5.3.

6 Conclusions

In this paper, several new number representations, based on a Residue Number System with the smallest prime numbers as moduli, were presented. Some of these representations, such as RPOS, turned out to be suitable for parallel computations on reconfigurable mesh architectures.

After developing all the techniques to convert numbers between all the representations commonly used on reconfigurable meshes, including the most important BIN $\to$ RPOS $\to$ BIN and BIN $\to$ RBIN $\to$ BIN, some applications are shown.

As a first application, an $O(1)$ time algorithm is shown for the Prefix Sums of N h-bit numbers on an $O\left(\frac{h^2+\log^2 N}{\log(h+\log N)}\right) \times O(N(h+\log N))$ reconfigurable mesh. This is the most efficient algorithm for reconfigurable meshes on the bit model. In addition to computing the Prefix Sums, this algorithm is also capable of reading the input and furnishing the output in constant time from the boundaries of the mesh. As a further application, the Approximate String Matching problem with α mismatches was considered on reconfigurable meshes for the first time, and solved in $O(1)$ time using an $O(m \log |\Sigma|) \times O\left(n\left(\log|\Sigma| + \frac{\log^2\alpha}{\log\log\alpha}\right)\right)$ reconfigurable mesh. The first results obtained in this paper with the new RNS representations on reconfigurable meshes are very promising. Therefore, it could be likely that further new and efficient algorithms for solving many other computational problems on reconfigurable meshes could be devised by using the techniques introduced in this paper.

Acknowledgments

This work was supported by grants from the University of Trento and the Provincia Autonoma di Trento.

References

[1] Y. Ben-Asher, D. Peleg, R. Ramaswani, and A. Schuster, "The Power of Reconfiguration," *J. Parallel and Distributed Computing,* vol. 13, no. 2, pp. 139-153, 1992.

[2] A.A. Bertossi and A. Mei, "P-Bandwidth Priority Queues on Reconfigurable Tree of Meshes," *J. Parallel and Distributed Computing,* vol. 40, pp. 248-255, 1997.

[3] A.A. Bertossi and A. Mei, "Constant Time Dynamic Programming on Directed Reconfigurable Networks," *IEEE Trans. Parallel and Distrubuted Systems,* vol. 11, no. 6, pp. 529-536, June. 2000.

[4] K. Bondalapati and V.K. Prasanna, "Reconfigurable Meshes: Theory and Practice," *Proc. Workshop Reconfigurable Architectures IPPS '97,* 1997.

[5] G.H. Chen, "An $O(1)$ Time Algorithm for String Matching," *Int'l. J. Computing Math.,* vol. 42, pp. 185-191, 1992.

[6] G.H. Chen and B.F. Wang, "Sorting and Computing Convex Hulls on Processor Arrays with Reconfigurable Bus Systems," *Information Sciences, An Int'l J.,* vol. 72, no. 3, pp. 191-206, 1993.

[7] K. Chung, "Image Template Matching on Reconfigurable Meshes," *Parallel Processing Letters,* vol. 6, no. 3, pp. 345-353, 1996.

[8] A. Frank, "Disjoint Paths in a Rectilinear Grid," *Combinatorica,* vol. 2, pp. 361-371, 1982.

[9] J. Jang, H. Park, and V.K. Prasanna, "An Optimal Multiplication Algorithm on Reconfigurable Mesh," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 5, pp. 521-532, May 1997.

[10] J. Jang and V.K. Prasanna, "An Optimal Sorting Algorithm on Reconfigurable Mesh," *J. Parallel and Distributed Computing,* vol. 25, pp. 31-41, 1995.

[11] J. Jang, V.K. Prasanna, and H. Park, "A Bit Model of Reconfigurable Mesh," *Proc. Reconfigurable Architectures Workshop IPPS '94,* Apr. 1994.

[12] J. Jenq and S. Sahni, "Reconfigurable Mesh Algorithms for Image Shrinking, Expanding, Clustering, and Template Matching," *Proc. Fifth Int'l Parallel Processing Symp.,* pp. 208-215, Apr. 1991.

[13] R. Lin, S. Olariu, J. Schwing, and J. Zhang, "A VLSI Optimal Constant Time Sorting on Reconfigurable Meshes," *Proc. Ninth European Workshop Parallel Computing,* 1992.

[14] R. Miller, V.K. Prasanna, D.I. Reisis, and Q.F. Stout, "Parallel Computations on Reconfigurable Meshes," *IEEE Trans. Computers,* vol. 42, no. 6, pp. 678-692, June 1993.

[15] K. Nakano, "Prefix-Sums Algorithms on Reconfigurable Meshes," *Parallel Processing Letters,* vol. 5, no. 1, pp. 23-35, 1995.

[16] K. Nakano and K. Wada, "Integer Summing Algorithms on Reconfigurable Meshes," *Theoretical Computer Science,* vol. 197, pp. 57-77, 1998.

[17] M. Nigam and S. Sahni, "Sorting n Numbers Using a $n \times n$ Reconfigurable Meshes with Buses," *Proc. Int'l Parallel Processing Symp.,* pp. 174-181, Apr. 1993.

[18] S. Olariu, J.L. Schwing, and J. Zhang, "Integer Problems on Reconfigurable Meshes, with Application," *J. Computer and Software Eng.,* vol. 1, no. 1, pp. 33-45, 1994.

[19] H. Park, H.J. Kim, and V.K. Prasanna, "An $O(1)$ Time Optimal Algorithm for Multiplying Matrices on Reconfigurable Meshes," *Information Processing Letters,* vol. 47, no. 2, pp. 109-113, 1993.

[20] Q.F. Stout, "Mesh Connected Computers with Broadcasting," *IEEE Trans. Computer,* vol. 3, no. 32, pp. 826-830, 1983.

[21] B.F. Wang, G.H. Chen, and F.C. Lin, "Constant Time Sorting on a Processor Array with a Reconfigurable Bus System," *Information Processing Letters,* vol. 34, pp. 187-192, 1990.

Alan A. Bertossi received the Laurea degree (summa cum laude) in computer science from the University of Pisa, Italy, in 1979. Afterward, he worked as a systems programmer and designer. From 1983-1994, he was with the Department of Computer Science, University of Pisa, as a research associate first and, later, as an associate professor. Since 1995, he has been with the Department of Mathematics, University of Trento, Italy, as a professor of computer science. His main research interests are the algorithmic aspects of high-performance, parallel, distributed, fault-tolerant, and real-time systems. He has published more than 50 refereed papers (two invited) in international journals, conferences, and encyclopedias. He is currently serving on the editorial board of *Information Processing Letters* and as guest coeditor for two special issues of *Algorithmica* and *Discrete Applied Mathematics*, both on experimental algorithmics. His biography is included in the edition of *Who's Who in the World* in 1999.

Alessandro Mei received the Laurea degree (summa cum laude) in computer science from the University of Pisa, Italy, in 1994, and the PhD degree in mathematics from the University of Trento, Italy, in 1999. Currently, he is a research fellow in the Department of Mathematics of the University of Trento. His research interest is the design and analysis of algorithms for parallel and reconfigurable systems.